Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Original) A method of operating a network file server for providing clients with concurrent write access to a file, the method comprising the network file server responding to a concurrent write request from a client by:

(a) obtaining a lock for the file; and then

(b) preallocating a metadata block for the file; and then

(c) releasing the lock for the file; and then

(d) asynchronously writing to the file; and then

(e) obtaining the lock for the file; and then

(f) committing the metadata block to the file; and then

(g) releasing the lock for the file.

2. (Previously presented) The method as claimed in claim 1, wherein the file further includes a hierarchy of blocks including an inode block of metadata, data blocks of file data, and indirect blocks of metadata, and wherein the metadata block for the file is an indirect block of metadata.

3. (Previously presented) The method as claimed in claim 2, which further includes copying data from an original indirect block of the file to the metadata block for the file, the original indirect block of the file having been shared between the file and a read-only version of the file.

4. (Previously presented) The method as claimed in claim 1, which further includes concurrent writing for more than one client to the metadata block for the file.

5. (Previously presented) The method as claimed in claim 1, wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing the partial write to the new block.

6. (Previously presented) The method as claimed in claim 1, wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the

conflict with the concurrent write to the new block, and then performing the partial write to the new block.

7. (Previously presented) The method as claimed in claim 6, which further includes placing a request for the partial write in a partial write wait queue upon finding an indication of a conflict with a concurrent write to the new block, and performing the partial write upon servicing the partial write wait queue.

8. (Previously presented) The method as claimed in claim 1, which further includes checking an input-output list for a conflicting prior concurrent access to the file, and upon finding a conflicting prior concurrent access to the file, suspending the asynchronous writing to the file until the conflicting prior concurrent access to the file is no longer conflicting.

9. (Previously presented) The method as claimed in claim 8, which further includes providing a sector-level granularity of byte range locking for concurrent write access to the file by the suspending of the asynchronous writing to the file until the conflicting prior concurrent access is no longer conflicting.

10. (Previously presented) The method as claimed in claim 1, which further includes writing the metadata block to a log in storage of the network file server for committing the metadata block for the file.

11. (Previously presented) The method as claimed in claim 1, which further includes gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

12. (Previously presented) The method as claimed in claim 1, which further includes checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file, and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on a staging queue for the file.

13. (Original) A method of operating a network file server for providing clients with concurrent write access to a file, the method comprising the network file server responding to a concurrent write request from a client by:

(a) preallocating a block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the block to the file;

wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method includes checking a partial block conflict queue for a conflict with a concurrent write to the new

block, and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block.

14. (Previously presented) The method as claimed in claim 13, wherein the method further includes placing a request for the partial write in a partial write wait queue upon finding an indication of a conflict with a concurrent write to the new block, and performing the partial write upon servicing the partial write wait queue.

15. (Original) A method of operating a network file server for providing clients with concurrent write access to a file, the method comprising the network file server responding to a concurrent write request from a client by:

(a) preallocating a metadata block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the metadata block to the file;

wherein the method includes gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining a lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

16. (Previously presented) The method as claimed in claim 15, which further includes checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the block to the file, and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on a staging queue for the file.

17. (Previously presented) The method as claimed in claim 15, wherein the network file server includes disk storage containing a file system, and a file system cache storing data of blocks of the file, and the method further includes the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache.

18. (Previously presented) The method as claimed in claim 17, which further includes the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale.

19. (Previously presented) The method as claimed in claim 18, which further includes the network file server checking a read-in-progress flag for a file block upon finding that the file

block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage, and then again checking whether the file block is in the file system cache.

20. (Previously presented) The method as claimed in claim 18, which further includes the network file server setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block.

21. (Previously presented) The method as claimed in claim 18, which further includes the network file server maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block.

22. (Previously presented) The method as claimed in claim 15, which further includes processing multiple concurrent read and write requests by pipelining the requests through a first

processor and a second processor, the first processor performing metadata management for the multiple concurrent read and write requests, and the second processor performing asynchronous reads and writes for the multiple concurrent read and write requests.

23. (Previously presented) The method as claimed in claim 15, which further includes serializing the reads by delaying access for each read to a block that is being written to by a prior, in-progress write until completion of the write to the block that is being written to by the prior, in-progress write.

24. (Previously presented) The method as claimed in claim 15, which further includes serializing the writes by delaying access for each write to a block that is being accessed by a prior, in-progress read or write until completion of the read or write to the block that is being accessed by the prior, in-progress read or write.

25. (Original) A method of operating a network file server for providing clients with concurrent read and write access to a file, the method comprising the network file server responding to a concurrent write request from a client by:

(a) preallocating a metadata block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the metadata block to the file;

wherein the network file server includes disk storage containing a file system, and a file system cache storing data of blocks of the file, and the method includes the network file server

responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache, and

which includes the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale.

26. (Previously presented) The method as claimed in claim 25, which further includes the network file server checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block, and then again checking whether the file block is in the file system cache.

27. (Previously presented) The method as claimed in claim 25, which further includes the network file server setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block.

28. (Previously presented) The method as claimed in claim 25, which further includes the network file server maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block.

Claims 29-31 (Cancelled).

32. (Original) A method of operating a network file server for providing clients with concurrent write access to a file, the method comprising the network file server responding to a concurrent write request from a client by executing a write thread, execution of the write thread including:

(a) obtaining an allocation mutex for the file; and then

(b) preallocating new metadata blocks that need to be allocated for writing to the file; and then

(c) releasing the allocation mutex for the file; and then

(d) issuing asynchronous write requests for writing to the file;

(e) waiting for callbacks indicating completion of the asynchronous write requests; and then

(f) obtaining the allocation mutex for the file, and then

(g) committing the preallocated metadata blocks, and then

(h) releasing the allocation mutex for the file.

33. (Original) A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) obtaining a lock for the file; and then

(b) preallocating a metadata block for the file; and then

(c) releasing the lock for the file; and then

(d) asynchronously writing to the file; and then

(e) obtaining the lock for the file; and then

(f) committing the metadata block to the file; and then

(g) releasing the lock for the file.

34. (Previously presented) The network file server as claimed in claim 33, wherein the file further includes a hierarchy of blocks including an inode block of metadata, data blocks of file data, and indirect blocks of metadata, and wherein the metadata block for the file is an indirect block of metadata.

35. (Previously presented) The network file server as claimed in claim 34, which is further programmed for copying data from an original indirect block of the file to the metadata block for the file, the original indirect block of the file having been shared between the file and a read-only version of the file.

36. (Previously presented) The network file server as claimed in claim 33, which is further programmed for concurrent writing for more than one client to the metadata block for the file.

37. (Previously presented) The network file server as claimed in claim 33, which further includes a partial block conflict queue for indicating a concurrent write to a new block that is being copied at least in part from an original block of the file, and wherein the network file server is further programmed to respond to a client request for a partial write to the new block by checking the partial block conflict queue for a conflict, and upon failing to find an indication of a conflict, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing a partial write to the new block.

38. (Previously presented) The network file server as claimed in claim 33, which further includes a partial block conflict queue for indicating a concurrent write to a new block that is being copied at least in part from an original block of the file, and wherein the network file server is further programmed to respond to a client request for a partial write to the new block by checking the partial block conflict queue for a conflict, and upon finding an indication of a

conflict, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block.

39. (Previously presented) The network file server as claimed in claim 38, which further includes a partial write wait queue, and wherein the network file server is further programmed for placing a request for the partial write in the partial write wait queue upon finding an indication of a conflict, and performing the partial write upon servicing the partial write wait queue.

40. (Previously presented) The network file server as claimed in claim 33, which is further programmed for maintaining an input-output list of concurrent reads and writes to the file, and when writing to the file, for checking the input-output list for a conflicting prior concurrent read or write access to the file, and upon finding a conflicting prior concurrent read or write access to the file, suspending the asynchronous writing to the file until the conflicting prior concurrent read or write access to the file is no longer conflicting.

41. (Previously presented) The network file server as claimed in claim 40, which is further programmed so that the suspending of the asynchronous writing to the file until the conflicting prior concurrent read or write access to the file is no longer conflicting provides a sector-level granularity of byte range locking for concurrent write access to the file.

42. (Previously presented) The network file server as claimed in claim 33, which is further programmed for maintaining an input-output list of concurrent reads and writes to the file, and when reading from the file, for checking the input-output list for a conflicting prior concurrent write access to the file, and upon finding a conflicting prior concurrent write access to the file, suspending the reading to the file until the conflicting prior concurrent write access to the file is no longer conflicting.

43. (Previously presented) The network file server as claimed in claim 42, which is further programmed so that the suspending of the reading to the file until the conflicting prior concurrent write access to the file is no longer conflicting provides a sector-level granularity of byte range locking for concurrent read access to the file.

44. (Previously presented) The network file server as claimed in claim 33, which is further programmed for committing the metadata block for the file by writing the metadata block to a log in the storage.

45. (Previously presented) The network file server as claimed in claim 33, which is further programmed for gathering together preallocated metadata blocks for a plurality of client requests for write access to the file, and committing together the preallocated metadata blocks for the plurality of client requests for access to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client requests for write access to the file, and then releasing the lock for the file.

46. (Previously presented) The network file server as claimed in claim 33, which further includes a staging queue for the file, and which is further programmed for checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file, and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on the staging queue for the file.

47. (Original) A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) preallocating a block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the block to the file;

wherein the network file server includes a partial block conflict queue for indicating a concurrent write to a new block that is being copied at least in part from an original block of the file, and wherein the network file server is programmed for responding to a client request for a partial write to the new block by checking the partial block conflict queue for a conflict, and upon finding an indication of a conflict, waiting until resolution of the conflict with the concurrent write to the new block of the file, and then performing the partial write to the new block of the file.

48. (Previously presented) The network file server as claimed in claim 47, which further includes a partial write wait queue, and wherein the network file server is programmed for placing a request for the partial write in the partial write wait queue upon finding an indication of a conflict, and performing the partial write upon servicing the partial write wait queue.

49. (Original) A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) preallocating a metadata block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the metadata block to the file;

wherein the network file server is programmed for gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining a lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

50. (Previously presented) The network file server as claimed in claim 49, which is further programmed for checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file,

and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on a staging queue for the file.

51. (Original) A network file server comprising disk storage containing a file system, and a file system cache storing data of blocks of a file in the file system, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) preallocating a metadata block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the metadata block to the file;

wherein the network file server is further programmed for responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache, and

wherein the network file server is programmed for responding to concurrent read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale.

52. (Previously presented) The network file server as claimed in claim 51, which is further programmed for checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a

prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block, and then again checking whether the file block is in the file system cache.

53. (Previously presented) The network file server as claimed in claim 51, which is further programmed for setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block.

54. (Previously presented) The network file server as claimed in claim 51, which is further programmed for maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block.

Claims 55-57 (Cancelled).

58. (Original) A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed with a write thread for responding to a concurrent write request from a client by:

(a) obtaining an allocation mutex for the file; and then

(b) preallocating new metadata blocks that need to be allocated for writing to the file; and then

(c) releasing the allocation mutex for the file; and then

(d) issuing asynchronous write requests for writing to the file;

(e) waiting for callbacks indicating completion of the asynchronous write requests; and then

(f) obtaining the allocation mutex for the file; and then

(g) committing the preallocated metadata blocks; and then

(h) releasing the allocation mutex for the file.

59. (Previously presented) The network file server as claimed in claim 58, which further includes an uncached write interface, a file system cache and a cached read-write interface, and wherein the uncached write interface bypasses the file system cache for sector-aligned write operations.

60. (Currently amended) The network file server as claimed in claim 59, wherein the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the ~~cached read-write~~ uncached write interface.

61. (Currently amended) A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) preallocating a block for writing to the file;

(b) asynchronously writing to the file; and then

(c) committing the preallocated block;

wherein the network file server also includes an uncached write interface, a file system cache, and a cached read-write interface, wherein the uncached write interface bypasses the file system cache for sector-aligned write operations, and the network file server is programmed to invalidate cache blocks in the file system cache including sectors being written to by the ~~cached read-write~~ uncached write interface.

62. (Previously presented) The method as claimed in claim 1, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

63. (Previously presented) The method as claimed in claim 13, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

64. (Previously presented) The method as claimed in claim 15, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

65. (Previously presented) The method as claimed in claim 25, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

66. (Previously presented) The method as claimed in claim 29, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

67. (Previously presented) The method as claimed in claim 32, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

68. (Previously presented) The method as claimed in claim 1, which further includes a final step of saving the file in disk storage of the network file server.

69. (Previously presented) The method as claimed in claim 13, which further includes a final step of saving the file in disk storage of the network file server.

70. (Previously presented) The method as claimed in claim 15, which further includes a final step of saving the file in disk storage of the network file server.

71. (Previously presented) The method as claimed in claim 25, which further includes a final step of saving the file in the disk storage.

72. (Previously presented) The method as claimed in claim 29, which further includes a final step of saving the file in disk storage of the network file server.

73. (Previously presented) The method as claimed in claim 32, which further includes a final step of saving the file in disk storage of the network file server.